

```

MostProfitableLoops (loopList, refList)
inputs : loopList ; // list of loops
           refList ; // list of references
output: profLoops ; // list of profitable loops

// Description from the paper: 'MostProfitableLoops returns
// the loop or set of loops that carries the most
// (unexploited) reuse.' Returns a list of loops ordered
// according to reuse type and amount carried by each loop.
// If two loops carry the same temporal reuse, a loop which
// also carries other types of reuse is more profitable.

profLoops ← Ø ;
foreach loop ∈ loopList do
    loop.retainedRefs ← Ø ; // refs w/ unexploited temporal reuse
    loop.partRetRefs ← Ø ; // refs w/ partially unexploited reuse
    loop.spatialRefs ← Ø ; // refs w/ spatial reuse
    loop.tempReuse ← 0 ;
    loop.partReuse ← 0 ;
    loop.spatialReuse ← 0 ;
    foreach ref ∈ refList do
        if ref.reuseType(loop) == TEMPORAL_REUSE then
            if ref is not yet mapped to any memory level then
                loop.retainedRefs ← loop.retainedRefs ∪ ref ;
                loop.tempReuse = loop.tempReuse + ref.reuseVal(loop) ;
            end
            else if ref is mapped to a single level then
                if ref is mapped to REGISTER_LEVEL then
                    loop.partRetRefs ← loop.partRetRefs ∪ ref ;
                    // this should account for reuse exploited in
                    // registers ... but reuse in registers
                    // depends on unroll sizes T.B.D. by search
                    loop.partReuse = loop.partReuse + ref.reuseVal(loop) ;
                end
            end
        end
        else if ref.reuseType(loop) == SPATIAL_REUSE then
            loop.spatialRefs ← loop.spatialRefs ∪ ref ;
            loop.spatialReuse = loop.spatialReuse + ref.reuseVal(loop) ;
        end
    end
    if loop.retainedRefs != Ø or
    loop.partRetRefs != Ø or
    loop.spatialRefs != Ø then
        profLoops.insert(loop) ;
    end
end
return profLoops ;

```